(4)

FINAL REPORT: COMPUTING ENVIRONMENTS FOR DATA ANALYSIS

by

John Alan McDonald

TECHNICAL REPORT No. 156

January 1989

Department of Statistics, GN-22

University of Washington

Seattle, Washington 98195 USA

Final Report
(1 Jan 1986 through 31 Dec 1988)
for
ONR Contract N00014-86-K-0069

A-1

Computing Environments for Data Analysis

JOHN ALAN MCDONALD
Dept. of Statistics, University of Washington

January 3, 1989

All work mentioned here was also supported (approximately 50%) by the *Nonparametric methods in multivariate analysis* project [July 1, 1985—July 29, 1988] funded by the Department of Energy Grant FG0685-ER25006.

1 Research Overview

The core of my proposal to ONR was the following sentence: *"The concrete goal of the proposed research is the design and implementation of a modern environment for interactive data analysis."* The extent to which I have succeeded in this goal is represented by the current state of a system called Arizona, which is described in detail in [13, 12]. Arizona is the latest in a series of "systems" which have been the focus of my research for the past three years. Each of the systems before Arizona investigated a particular aspect of a comprehensive data analysis environment; the purpose of Arizona is to put them together. In brief, the systems were:

- Antelope: experimentation with object-oriented databases and constraint-based interactive graphics.

- Fossil: interfaces to traditional (Fortran) scientific subroutine libraries.
- Cactus: a mathematician's numerical linear algebra package, using object-oriented programming to provide a higher level of abstraction than Linpack[6].

1.1 Arizona

Arizona is intended to be a portable, public-domain collection of tools supporting scientific computing, quantitative graphics, and data analysis, implemented in Common Lisp[22, 21, 7] and CLOS (the Common Lisp Object System)[3, 10]. Although there is substantial implementation of some of the modules described below, more years of work are required before it will mature and stabilize to the point of robust production-quality code. Arizona is intended primarily as a research vehicle; however, I hope that the ideas embodied in its design are of interest in themselves and of use in future scientific computing and data analysis systems (eg. a "New New S"[2]).

Arizona evolved out of the Fossil meeting discussed below. The code in the current version of Arizona is perhaps 80% my work, with the remainder contributed by John Michalak, Michael Sannella, Werner Stuetzle, Robert Gentleman, Catherine Hurley, and Andrew Bruce of the U. of Washington, Jan Pedersen of Stanford University and Xerox PARC, Steve Peters of MIT, UC Berkeley, and Apple Computers, and Wayne Oldford of MIT and U. of Waterloo. In addition, code from Arizona has been distributed to and is the subject of continuing collaboration with researchers at U. Minnesota, Carnegie Mellon, Bell Labs, and Bellcore.

Discussion of the philosophy underlying Arizona can be found in [14, 15, 11, 12, 16, 23]. Briefly, the design is motivated by the belief that an ideal system for scientific computing and data analysis should have:

- One language that can be used for both line-by-line interaction and defining compiled procedures.
- Minimal overhead in adding new compiled procedures (or other definitions).

- A language that supports a wide variety of abstractions and the definition of new kinds of abstractions.
- Programming tools (editor, debugger, browsers, metering and monitoring tools).
- Automatic memory management (dynamic space allocation and garbage collection).
- Portability over many types of workstations and operating systems.
- A community of users and developers.
- Access to traditional Fortran scientific subroutine libraries or equivalents.
- A representation of scientific data directly in the data structures of the language.
- Comprehensive numerical, graphical, and statistical functionality.
- Device independent static output graphics.
- Window based interactive graphics.
- Support for efficient and concurrent access to large databases.
- Documentation and tutorials, both paper and on-line.

The first nine points (through "access to Fortran") come for free with standard Common Lisp environments. The remaining six are the research aspects of Arizona.

Arizona is divided into a number of modules, with limited interdependencies, to permit individual modules to stabilize and be "released" before the whole system is complete.

The modules are divided into two groups: a numerical, quantitative kernel and an interactive, window-based, scientific graphics part.

1.1.1 A quantitative kernel

The non-graphical quantitative kernel is more developed at present, because it can be implemented in an efficient, portable way using existing standards for Common Lisp and CLOS. The quantitative kernel consists of:

- *Basic Math,* which requires Common Lisp,
- *Collections,* which requires Common Lisp and CLOS,
- *Linear Algebra,* which requires Basic Math and Collections (sometimes this module is refered to as *Cactus*),
- *Probability,* which requires Linear Algebra,
- *Database,* which requires Collections, and
- *Statistics,* which requires Database and Probability.

The contents of these modules are described in [13]. The Linear Algebra module is the Cactus system discussed below.

1.1.2 Static output-only graphics

Since [13] was written, static, output-only graphics has been added to Arizona using two modules, which are based on the Xerox Lisp PLOT package designed and written by Jan Pedersen[17]:

- *2D-Graphics,* which is essentially a device driver for various Common Lisps and displays. It defines a Graphics-Canvas abstraction which provides the ability to draw points, lines, polylines, bitmaps, characters, strings, etc., in device coordinates. Implementations exist for Xerox Common Lisp, Symbolics Genera Windows, and Coral Common Lisp (on the Apple Mac II). Implementations for X windows and Postscript printers are planned for the near future.

- *2D-Plot*, which provides higher-level scientific graphics, such as scatterplots, histograms, boxplots, etc. The basic abstraction is the Plot, which is a hierarchy of Plot-Objects (points, curves, strings, etc.) that are defined in the user or world coordinate system. A Plot can be drawn on any available Graphics-Canvas (eg. a window or a printer); the transformation from world to device coordinates is hidden from the user. It is fairly easy for the user to define new types of plots by creating new hierarchies of Plot-Objects.

1.1.3 Interactive, Motion Graphics

The current design for interactive graphics is fairly tentative. Implementation of a portable scientific graphics toolkit requires a standardized interface between Common Lisp/CLOS and the large variety of proprietary or proposed standard window systems for workstations and personal computers (eg. Symbolics Genera [25], NeWS[24], X[20], etc.). This standard (sometimes called Common Windows) is the subject of intense activity in the Common Lisp community[9, 18]. 2D-Graphics/2D-Plot is our current best approximation; it will be modified or will wither away as a true Common Windows standard emerges.

The parts of 2D-Graphics and 2D-Plot that are not specific to a particular display device were implemented in pure Common Lisp, for portablility. Since the CLOS standard is now more or less fixed, we will convert them to use it, as a first step in supporting interactive graphics.

I have identified three modules in a future interactive graphics subsystem:

- *Constraints*, which requires Common Lisp and CLOS. (This module might very well be part of the non-graphical kernel, but most of the applications we have in mind at present are in graphics.)
- *Quantitative Graphics*, which requires Common Windows, Collections, Constraints, and Linear Algebra.
- *Data Analysis Graphics*, which requires Quantitative Graphics and Statistics

The contents of these modules are described in [13].

1.2 Cactus

The underlying premise of much of my research is that quantitative, scientific computing needs and deserves good programming languages and programming environments—as much as artificial intelligence. The Cactus system shows how straightforward application of object-oriented design to standard algorithms in numerical analysis yields immense improvement in clarity, without sacrificing speed or numerical accuracy.

Object-oriented programming is sometimes said to only be useful for graphics and user interface and, perhaps, knowledge representation and database management. Even Lisp machine manufacturers seem to think that Fortran is somehow superior for traditional, quantitative scientific computing. When numerical software is written in Lisp (eg. [19]), the author usually adopts a Fortran or Algol style and neglects the potential for designing more appropriate abstractions[1].

Cactus is based on a collection of *linear transformation* classes and appropriate generic operations. This level of abstraction greatly simplifies many algorithms in numerical linear algebra. Traditional linear algebra systems (Linpack[6], APL) operate at the level of arrays and confound the details of where data is kept with how it is meant to be used.

1.3 Fossil

I organized a small meeting in Seattle in November, 1986, which was attended by:

- Keith Kerr, Applied Physics Laboratory, U. Washington
- John McDonald, Statistics, U. Washington
- John Michalak, Statistics, U. Washington
- Wayne Oldford, U. Waterloo
- Jan Pedersen, Xerox and Stanford
- Stephen C. Peters, U.C. Berkeley

- Werner Stuetzle, Statistics, U. Washington
- Alan Wilks, AT&T Bell Labs.

The purpose of the meeting was to collect together a group of people interested in statistical computing in Lisp. We intended to collaborate in an informal joint project called *Fossil*. The primary goal of *Fossil* was to build up a portable library of scientific subroutines—statistical, numerical, and graphics functions—that can be shared by people working in any implementation of Common Lisp. This would eliminate a present tendency towards re-inventing the wheel among statistics researchers using Common Lisp. To be able to share software, we needed to agree on a number of things, in particular, consistant data structures and function libraries.

The most direct, concrete result of the Fossil meeting was Polish-Fortran, a portable Common Lisp program that translates Fortran source code into a Lisp-like syntax (polish notation). A set of portable Common Lisp macros are provided that simulate Fortran semantics. Although some Lisp environments provide mechanisms for calling Fortran subroutines, these foreign function interfaces are not standardized and, therefore, code that relies on them is not portable. In addition, the translator can and has been used as a first step in translating Fortran into readable, modifiable, extensible Common Lisp code. This was done for some of the special functions in the Basic Math module in Arizona.

1.4 Antelope

One part of my research was devoted to two related, "high level" topics: object-oriented representation of statistical data and *Object-viewing*, a constraint approach to data analysis graphics. The result of this work is an experimental system called *Antelope*, which is described in [11].

An important part of Antelope was the Antelope language. I extended the object-oriented Flavors language provided on the Symbolics, because it did have all the properties necessary to support the statistical object-oriented databases or constraint-based graphics. The Antelope language is now unnecessary, because CLOS supplies some of the features that were

missing in Flavors, and provides a systematic and portable way to add others via the meta-object protocol.

In Antelope, statistical data is represented by objects, the primitive data structures in the object-oriented Antelope language. The choice to represent statistical data directly in the most primitive data structures of the programming language, rather than adding a more elaborate statistical database, is both minimal and harmonious with the spirit of base environment. The relevant point here is that statistical data objects do not differ in any fundamental way from other objects in the programming environment. Statistical tools can be applied to any objects (for example, the windows on the screen or the processes managed by the scheduler); the regular programming tools (for example, the editor or the inspector) can be used on statistical objects.

The constraint paradigm for graphics is used to support *Object-viewing*, which means that windows are thought of as views of the current state of objects in the environment. For example, scatterplot windows are graphical views of data sets; editor windows are text views of procedure objects. The key point is that windows automatically update their appearance whenever the objects they show change state. If a record object has a slot called `color`, and the value of `color` is changed from **red** to **green**, then all windows that show some representation of that particular record will change the color from red to green. The implementor of code which modifies a record object does not have to worry about whether the record is shown in one or more windows and that these windows need to be updated whenever the record changes.

In contrast, most existing window systems treat windows as virtual paper and their contents as virtual ink. Drawing on a window makes a mark that remains until it is explicitly erased and re-drawn. One motivation for a systematic implementation of object-viewing is the belief that most erasing and redrawing of windows is done to reflect a change of state of some object in the environment. Thus common tasks in interactive graphics are simplified if erasing and redrawing can be done automatically.

Object-viewing is best thought of as imposing a constraint between the abstract object in the programming environment and its visible representation on the screen. A good constraint language gives the programmer clear, simple ways to express constraints and also encourages a modular design—

by separating the description and implementation of constraints from the implementation of the internal details of the constrained objects.

In a little more detail, the object-viewing programming model consists of:

- A number of *base objects* residing in the programming environment's address space, like a record in a statistical database or the definition of a Lisp function.
- *Visible representations* of base objects in windows. Abstractly, the visible representation corresponds to something like a node of a hierarchical display tree.
- A *viewing filter* (or viewing pipeline [5]) that computes the state of the visible representation from the state of the base object. An obvious example of a viewing filter is the usual 3-d viewing transformation, which might be used is a rotating scatterplot. A less obvious example is a pretty-printer that is applied to the definition of a Lisp function before it is displayed in an editor window.
- A *constraint* that causes the visible representation to be re-computed (and re-drawn) whenever the state of the base object changes.

Further applications of the constraint paradigm to statistical graphics are described in [4, 5, 8].

2 Professional Activity

2.1 Refereed Publications:

1987 *Computing Environments for Data Analysis, Part III: Programming Environments,* with Jan Pedersen, SIAM J. Scientific and Statistical Computing 9 (2): 380-400.

1986 *Smoothing with Split Linear Fits,* with Art Owen, Technometrics 28 (3): 195-208.

1986 *Periodic Smoothing of Time Series,* SIAM J. Scientific and Statistical Computing 7 (2): 665-688.

2.2 Invited Papers in Conference Proceedings:

1988 *An outline of Arizona: a portable Lisp-based system for data analysis,* at Computer Science and Statistics, the 20th Symposium on the Interface, Reston, Va., April. (Also Tech Rept 131, Dept. of Statistics, U. of Washington)

1986 *Antelope: data analysis with object-oriented programming and constraints,* Proc. of the 1986 Joint Statistical Meetings, Stat. Comp. Sect. (also Tech Rept 89, Dept. of Statistics, U. of Washington).

2.3 Other Publications:

1988 *Elements of a viewing pipeline for data analysis,* with Andreas Buja, Daniel Asimov, and Catherine Hurley, in Dynamic Graphics for Statistics, Cleveland, W.S., and McGill, M.E., eds. (1988) Wadsworth, Pacific Grove, Ca.

1988 *Interactive Graphics for Data Analysis,* (Ph.D. Thesis) Available as Tech. Report Orion# 11, Dept. of Statistics, Stanford University. Also in Dynamic Graphics for Statistics, Cleveland, W.S., and McGill, M.E., eds. (1988) Wadsworth, Pacific Grove, Ca.

1987 *Object-oriented design in numerical linear algebra,* Technical Report 109, Dept. of Statistics, U. of Washington.

2.4 Invited Presentations:

1988 *Analysis of fish abundance in the Bering sea: a case study in the use of graphical methods,* with Werner Stuetzle, at the Joint Statistical Meetings in New Orleans in August 1988.

1988 *An outline of Arizona: a portable Lisp-based system for data analysis,* at Computer Science and Statistics, the 20th Symposium on the Interface, Reston, Va., April.

1987 *Object-oriented design in numerical linear algebra,* at Computer Science and Statistics, the 19th Symposium on the Interface, Philadelphia, March.

1986 *Antelope: data analysis with object-oriented programming and constraints,* Proc. of the 1986 Joint Statistical Meetings, Stat. Comp. Sect.

1986 *The object-viewer paradigm for data analysis graphics* at the AMS meeting on Data Analysis and Graphics in Santa Cruz in June.

1986 *The object-viewer paradigm for data analysis graphics* at the session on Graphical Components for Statistical Workstations of the National Computer Graphics Association meeting in May.

I have given invited seminars in the Statistics Dept., Computer Science Dept., the Applied Physics Lab at the U. of Washington, at Stanford, MIT, U. of Waterloo, and Florida State, and in research labs at AT&T Bell Labs, Bellcore, Schlumberger-Doll Research, Schlumberger Palo Alto, and Xerox PARC.

In addition, the ONR Young Investigator Award has supported my attendance at a number of conferences where I did not give presentations: the Common Lisp Object System Workshop at Xerox PARC in October 1988, the 2nd OOPSLA (Object-Oriented Programming: Systems, Languages, and Applications) meeting in Orlando in October 1987, the Symbolics National Users Group meeting in Seattle in July 1987, the AAAI meeting in Seattle in July 1987, the Joint Statistical Meetings in San Francisco in August 1987, and the 1st OOPSLA meeting in Portland in October 1986.

2.5 Graduate Students

- Ph.D. committee for Catherine Hurley. Degree received Fall 1987. Thesis title: *The Data Viewer: A program for graphical data analysis.*
- Ph.D. committee for Jeff Banfield. Degree received spring 1988. Thesis topic: *Clustering and image processing with applications to remote sensing of sea ice.*
- Ph.D. committee for Pat Burns. Degree received in Jan 1988. Thesis topic: *Resistant Fits of Two-way tables.*
- Ph.D. committee for John Michalak. Degree to be received spring 1989. Thesis topic: *Interactive methods for viewing hierarchical structures in data analysis.*

- Ph.D. committee for Charlie Geyer. Thesis topic: *Constrained Maximum Likelihood for exponential families.*

In addition, I informally advise other Statistics, Biostatistics, and Computer Science Ph.D. students who use our Lisp machines in their research, including Deborah Donnell, Andrew Bruce, Robert Gentleman, Mike Kahn, Jorean Sicks, Steve McKinney (Biostat), Kevin Anderson (Biostat), Jed Dennis, Suzanne Weghorst (CS), and Michael Sannella (CS).

References

[1] H. Abelson, G. Sussman, and J. Sussman. *Structure and Interpretation of Computer Programs.* MIT Press, Cambridge, Mass., 1985.

[2] R.A. Becker, J.M. Chambers, and A.R. Wilks. *The New S Language.* Wadsworth and Brooks/Cole, Pacific Grove, CA, 1988.

[3] D.G. Bobrow, L.G. DeMichiel, R.P. Gabriel, S. Keene, G. Kiczales, and D.A. Moon. *Common Lisp Object System Specification X3J13 Document 88-002R*, 1988.

[4] A. Buja, C. Hurley, and J.A. McDonald. A data viewer for multivariate data. In *Computer Science and Statistics: Proc. 18th Symp. on the Interface*, Washington, D.C., 1987. ASA.

[5] A. Buja, C. Hurley, and J.A. McDonald. Elements of a viewing pipeline for data analysis. In W.S. Cleveland and M.E. McGill, editors, *Dynamic Graphics for Statistics.* Wadsworth and Brooks/Cole, Belmont, Ca., 1987.

[6] J.J. Dongarra, C.B. Moler, J.R. Bunch, and G.W. Stewart. *LINPACK Users' Guide.* SIAM, Philiadelphia, 1979.

[7] Franz, Inc. *Common Lisp: the reference.* Addison-Wesley, Reading, Mass., 1988.

[8] C. Hurley. *The data viewer: a program for graphical data analysis.* PhD thesis, Dept. of Statistics, U. of Washington, 1987.

[9] Intellicorp. *Intellicorp Common Windows Manual.* 1975 El Camino Real West, Mountain View, Ca 94040-2216, 1986.

[10] Sonya E. Keene. *Object-oriented programming in Common Lisp: a programmer's guide to CLOS.* Symbolics Press and Addison-Wesley, Reading, Mass., 1388.

[11] J.A. McDonald. Antelope: data analysis with object-oriented programming and constraints. In *Proc. of the 1986 Joint Statistical Meetings, Stat. Comp. Sect.*, 1986. Also Tech Rept 89, Dept. of Statistics, U. of Washington.

[12] J.A. McDonald. Object-oriented design in numerical linear algebra. Technical Report 109, Dept. of Statistics, U. of Washington, 1987.

[13] J.A. McDonald. An outline of arizona. Technical Report 131, Dept. of Statistics, U. of Washington, 1988.

[14] J.A. McDonald and J.O. Pedersen. Computing environments for data analysis I: Introduction. *SISSC*, 6(4):1004–1012, 1985.

[15] J.A. McDonald and J.O. Pedersen. Computing environments for data analysis II: Hardware. *SISSC*, 6(4):1013–1021, 1985.

[16] J.A. McDonald and J.O. Pedersen. Computing environments for data analysis III: Programming environments. *SISSC*, 9(2):380–400, 1988.

[17] Jan Otto Pedersen. *IDL—A statistical programming environment.* PhD thesis, Dept. of Statistics, Stanford University, 1989.

[18] R.B. Rao. Towards interoperability and extensibility in window environments via object-oriented programming. Master's thesis, MIT EECS, 1987.

[19] G. Roylance. Some scientific subroutines in lisp. Technical Report Memo 774, MIT AI Lab, 1984.

[20] R.W. Scheifler and J. Gettys. The X window system. *ACM TOG*, 5(2):79–109, 1986.

[21] Rosemary Simpson. *Common Lisp: the index.* Coral Software, Inc. and Franz, Inc., 707 Laurel St., Menlo Park, Ca. 94025, 1987.

[22] G.L. Steele. *Common Lisp, The Language.* Digital Press, 1984.

[23] W. Stuetzle. Plot windows. *JASA*, 82(398):466–475, 1987.

[24] Sun Microsystems, Inc. *NeWS Manual.* Sun Microsystems, Inc., 2550 Garcia Ave, Mountain View, Ca. 94043, 1987. Part No. 800-1632-10.

[25] J. Walker, D.A. Moon, D.L. Weinreb, and M. Macmahon. The Symbolics Genera programming environment. *IEEE Software*, 4(6):36–45, 1987.